#35 LDJ 5-19-04

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

OFFICIAL

Applicant(s):	Marc Tremblay and William N. Joy		
Title:	IMPLICITLY DERIVED REGISTER SPECIFIERS IN A PROCESSOR		
Application No.:	09/204,479	Filed:	December 3, 1998
Examiner:	David Y. Eng	Group Art Unit:	2155
Atty. Docket No.:	004-3289		

May 10, 2004

Mail Stop - Appeal Briefs - Patents
COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, VA 22313-1450

APPELLANT'S REQUEST FOR ORAL HEARING (37 C.F.R. § 1.194)

Appellant hereby requests an Oral Hearing in the above-referenced Appeal. The fees required under 37 C.F.R. § 1.17(d) are provided in the accompanying Transmittal.

CERTIFICATE OF MAILING OR TRANSMISSION

I hereby certify that, on the date shown below, this correspondence is being

☐ deposited with the US Postal Service with sufficient postage as first class mail, in an envelope addressed to Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

☒ facsimile transmitted to the US Patent and Trademark Office.

David W. O'Brien — May 10, 2004 (Date)

EXPRESS MAIL LABEL:

Respectfully submitted,

David W. O'Brien, Reg. No. 40,107
Attorney for Applicant(s)
(512) 338-6314
(512) 338-6301 (fax)

05/20/2004 LJOHNSON 00000001 500631 09204479
01 FC:1403 290.00 DA

3. If K = N, the algorithm terminates and the answer is on the stack. Otherwise add 1 to K and go to step 2.

Figure 3-29 shows the evaluation of the same formula as in Fig. 3-28 but using a stack this time. Note that the number on top of the stack is the right operand, not the left operand.

A computer organized around a stack offers several advantages compared to multiregister machines, such as the 370, Cyber 70, and PDP-11:

1. Short instructions, because many instructions have no addresses.
2. Formulas are easy to evaluate.
3. There is no need for complicated algorithms to optimize register use.

Although neither the 370, the Cyber 70, nor the PDP-11 have stack arithmetic instructions, some compilers for these machines use the registers to simulate a stack in software, at a certain cost in performance compared to having level 2 instructions to perform stack arithmetic directly.

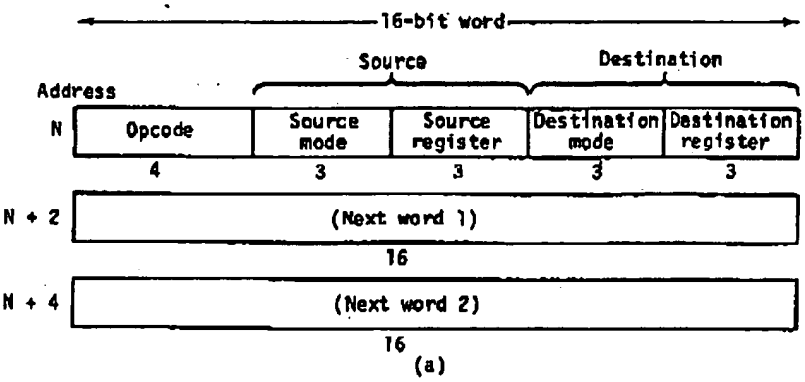
Although the PDP-11 does not have stack arithmetic instructions, it does have stack addressing. Register 6 is the main stack pointer, although other registers may be used as stack pointers for auxilliary stacks. The instructions to push a word onto a stack and pop a word off a stack use autoindexing on the stack pointer.

3.3.8. Addressing on the PDP-11

The previous sections discussed a number of different addressing modes—immediate addressing, direct addressing, indirect addressing, indexing, and so on. There still remains the question of how the hardware or level 1 interpreter knows whether an address is immediate, direct, indirect, indexed, and so on. One solution is to have a separate opcode for each addressing mode—that is, separate opcodes for ADD IMMEDIATE, ADD DIRECT, ADD INDIRECT, and so on. Another way is to make the mode part of the address. Each instruction could contain a few bits per address specifying which form of addressing was desired.

The PDP-11 has the largest number of addressing modes of the level 2 machines we have examined and is worth examining in detail to see how the addressing works. Figure 3-30(a) shows the format of the two-operand instructions, such as MOVE, ADD, COMPARE, and INCLUSIVE OR. Each operand has a three-bit addressing mode and a three-bit register field. The meaning of the modes is given in Fig. 3-30(b). Remember that register 7 is the program counter and that it is advanced by 2 immediately after an instruction word is fetched (before the instruction is executed).

All PDP-11 instructions are actually only 16 bits, but, in some cases, one or two extra words directly following the instruction are used by the instruction and can be considered part of it. Mode 6 and mode 7 addressing require a 16-bit constant for indexing. Furthermore, if either mode 2 or mode 3 is specified with register 7 (the program counter), the following sequence of steps occurs.



(a)

Mode	Name	How the operand is located
0	Register addressing	The operand is in R.
1	Register indirect	R contains a pointer to the operand.
2	Autoincrement	The content of R is fetched and used as a pointer to the operand. After this step, but before the instruction is executed, R is incremented by 1 (byte instructions) or 2 (word instructions).
3	Autoincrement indirect	The address of a memory word containing a pointer to the operand is fetched from R. Then R is incremented by 1 or 2 before the instruction is executed.
4	Autodecrement	R is first decremented by 1 or 2. The new value of R is then used as a pointer to the operand.
5	Autodecrement indirect	R is first decremented by 1 or 2. The new value of R is then used as the address of a memory location containing a pointer to the operand.
6	Indexing	The operand is at the address equal to the sum of R (the index register) and the 16 bit 2's complement offset in the next word. In modes 6 and 7, the program counter (R7) is incremented by 2 immediately after the next word is fetched.
7	Indexing + indirect addressing	The memory location containing a pointer to the operand is found by adding the contents of R and the next word. In modes 6 and 7, the program counter (R7) is incremented by 2 immediately after the next word is fetched.

(b)

Fig. 3-30. (a) Format of a PDP-11 2-address instruction. (b) Description of the PDP-11 addressing modes. *R* is the register specified along with the mode.

First, the instruction is fetched and register 7 is increased by 2 (a word is two bytes). Then register 7 is used as a pointer to the data (mode 2) or the address of the data (mode 3). In both cases, the word pointed to is the word following the instruction. After this word is fetched, the autoincrementing of register 7 takes place, increasing register 7 by 2. Autoincrement addressing that specifies the program counter is a clever trick allowing the word following the instruction to be used as data. In mode 2 this data is the operand, yielding immediate addressing. In mode 3 this data is the address of the operand, yielding direct addressing. If both source and destination require an extra word, the first one is for the source.

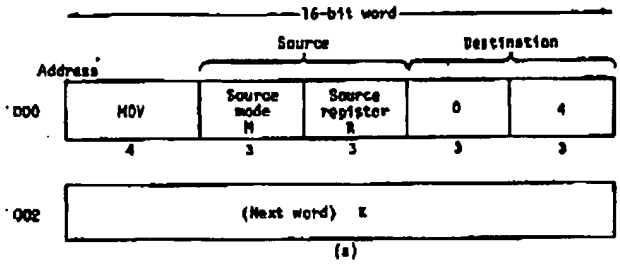
The PDP-11 has an interesting form of addressing called **self-relative** or **position-independent** addressing. When mode 6 and register 7 are specified, the operand address is found by forming the sum of the index word following the instruction and the program counter. In effect, the index word gives the operand address by specifying how far away it is from the instruction itself, either in front or behind.

If all memory references use this form of addressing instead of direct addressing (mode 3 with register 7), a program can be loaded anywhere in memory and it will run correctly. In addition, it can also be moved after being loaded, because although the absolute addresses of the needed operands change, their distance from the instructions referencing them remains fixed.

As an example of the power of the PDP-11 addressing mechanism, consider the MOV instruction of Fig. 3-31(a). This instruction moves the source operand to register 4. Figure 3-31(b) shows all the different variations of this instruction for different source modes and registers.

Because both the source addressing mode and the destination addressing mode can be independently specified, a single opcode yields a large number of different instructions. For example, the ADD instruction can be used to

add a register to another register (0, 0)
add a register to a memory word (0, 6)
add a memory word to a register (6, 0)
add a memory word to another memory word (6,6)
pop a word from a stack and add it to a register (2, 0)
pop a word from a stack and add it to a memory word (2, 6)
add an immediate operand to a register (2, 0)
add an immediate operand to a memory word (2, 6)
add an immediate operand to the top word on the stack (2, 1)
add a register to the top word on the stack (0, 1)
add a memory word to the top word on the stack (6, 1)
add a memory word to an indirectly specified address (6, 7)



(a)

Source register

Source mode M	R = 0 - 5	R = 6	R = 7
0	Move R to R4 (register - register) Example: MOV R3, R4	Move stack pointer to R4 Example: MOV SP, R4	Move program counter to R4 Example: MOV PC, R4
1	Move memory word pointed to by R to R4 Example: MOV @R3, R4	Move top of the stack to R4, but do not remove it from the stack Example: MOV @SP, R4	Move K to R4; program counter is not incremented again so K will be executed as the next instruction Example: MOV @PC, R4
2	Move memory word pointed to by R to R4 and add 2 to R Example: MOV (R3)+, R4	Remove a word from the stack and put it in R4 (pop instruction) Example: MOV (SP)+, R4	Move K to R4 (immediate) addressing) Example: MOV #24, R4
3	Move to R4 the memory word addressed by the word R points to, and add 2 to R Example: MOV @(R3)+, R4	Pop the address of the source operand from the stack, and move the source operand itself to R4 Example: MOV @(SP)+, R4	Load R4 from memory address K (direct addressing) Example: MOV @ #24, R2
4	Decrement R by 2 and then load R4 from the address R points to Example: MOV -(R3), R4	M = 4 and R = 6 is not useful as a source; however it is used as a destination in push instructions Example of push: MOV #6, -(SP)	Not used (causes an infinite loop)
5	Decrement R by 2 and then load R4 indirectly from the address R points to Example: MOV @-(R3), R4	Not used	Not used (causes an infinite loop)
6	Load R4 with the memory word at C(R) + K (indexing) Example: MOV 24(R3), R4	Load R4 with the word K/2 words below the top of the stack Example: MOV 24(SP), R4	Load R4 with the word K/2 words from this instruction (self relative addressing) Example: MOV X, R4 Note: The assembler computes the appropriate constant to address X
7	Load R4 with the memory word pointed to by C(R) + K (indexing + indirect addressing) Example: MOV @24(R3), R4	Load R4 from the word whose address is K/2 words below the top of the stack Example: MOV @24(SP), R4	Load R4 with the word pointed to by the word K bytes from this instruction (indirect addressing) Example: MOV @X, R4 Note: The assembler computes the appropriate constant to address X

(b)

Fig. 3-31. (a) A PDP-11 instruction that moves a word to register R4. (b) Different variations of this instruction for different source modes and registers. *R* is the register and *C(R)* is its contents. The assembly language notation is shown in the examples.

add a register to an indirectly specified address (0, 7)

add an immediate operand to an indirectly specified address (2, 7)

as well as numerous other possibilities. The numbers in parentheses are the source and destination modes. Note that mode 6 can always be replaced by mode 3; that is, memory can be addressed self-relative or directly. Also note that mode 1 with register 6 uses the top word of the stack as a source or destination but does not remove it from the stack, whereas mode 2 with register 6 does remove it.

Although only one opcode is used, the PDP-11 ADD instruction has dozens of distinct and useful variations. If all the variations of all 12 two-address instructions were counted as separate instructions, the PDP-11 would have hundreds of instructions. Because of the extreme flexibility of its addressing modes, the PDP-11 has a far more powerful and useful instruction set than the instruction list might indicate at first glance.

3.3.9. Discussion of Addressing Modes

In contrast to the PDP-11, the 370 and Cyber 70 (which were designed years before the PDP-11) lack indirect addressing, autoindexing, and stack addressing. Furthermore, the addressing modes of their operands are not independently specifiable. For example, the 370 has different opcodes for add memory to register and add register to register, and no instructions to add a register to a memory word or one memory word to another. (There is an instruction to add two numbers in memory in decimal format, however.)

The freedom of a computer designer to include or exclude certain types of addressing from a new machine is severely constrained by the number of bits available for encoding the addressing information. For instance, let us consider why the Cyber 70 does not have RX (register to memory, indexed) instructions, like the 370. These instructions require a memory address and two registers—one for the data and one for the indexing.

On the Cyber 70, a memory address uses 18 bits (although 17 would suffice) and a register 3 bits, making 24 bits for a memory address and two registers. A 30-bit instruction would leave only six bits over for the opcode. RX instructions for addition, subtraction, multiplication, or division of integers or floating-point numbers would each require a separate opcode, and as we have seen, spare opcodes are in short supply on the Cyber 70.

As another example, consider why the IBM 370 does not have three-register addressing as the Cyber 70 does. Adding a third four-bit register field to the RR instructions would have required extending those instructions to 20 bits, hardly a reasonable value considering that all the other instructions lengths are multiples of a half word. The RR instructions could have been extended to 32 bits, but doing so would have been very wasteful of space, unless a use could have been found for the remaining 12 bits.

Moreover, allowing three registers in the RR format (which would probably have